Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 1 (2012) 2165–2174

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Computational Science, ICCS 2010

Automatic Design Optimization using Parallel Workflows

David Abramson[a*], Blair Bethwaite[a], Colin Enticott[a], Slavisa Garic[a], Tom Peachey[a], Anushka Michailova[b], Saleh Amirriazi[b]

[a]*Faculty of Information Technology, Monash University, Clayton, 3800, Victoria, Australia*
[b]*Department of Bioengineering, University of California San Diego, 9500 Gilman Drive, La Jolla, USA*

Abstract

Workflows support the automation of scientific processes, leading to a more robust experimental process. They facilitate access to remote instruments, databases and parallel and distributed computers. Importantly, software pipelines can be established that perform multiple complex simulations (leveraging distributed platforms), with one simulation driving another. Such an environment is ideal for performing engineering design, where the goal is to evaluate a range of different scenarios "in-silico", and find ones that optimize a particular outcome. However, in general, existing workflow tools do not incorporate optimization algorithms, and thus whilst users can specify complex computational and data manipulation pipelines, they need to invoke the workflow as a stand-alone computation within an external optimization tool. Moreover, many existing workflow engines do not leverage parallel and distributed computers, making them unsuitable for executing complex engineering simulations. To solve this problem, we have developed a methodology for integrating optimization algorithms directly into workflows. We implement a range of generic actors for an existing workflow system called Kepler, and discuss how they can be combined in flexible ways to support various different design strategies. We illustrate the system by applying it to an existing bio-engineering design problem running on a Grid of distributed clusters.

Keywords: Design optimization, Grid computing, Cardiac models, workflows ;

1. Introduction

Scientific workflows are evolving as a platform for automating both experimental and “in-silico” science. These make it possible to specify experiments involving a range of different real and computational activities, such as instrument control, data integration, modelling and analysis, and visualization. Workflows are sometimes created using a graphical programming environment. The output of one activity can be passed as input to the next, forming a pipeline of arbitrary complexity. Some workflow engines manage the execution across a range of distributed resources, and leverage Grid computing middleware and approaches where appropriate to control and interact with

* Corresponding author. Tel.: +61-3-99051183.
E-mail address: david.abramson@infotech.monash.edu.au

1877-0509
doi:10.1016/j.procs.2010.04.242

remote resources. An enormous number of workflow engines have been built, each with different characteristics and features. A good review can be found in [27].

Workflows are particularly useful for in-silico design, because they make it possible to perform a number of different computation steps (possibly on different computers) and to build a pipeline that models a complex design process. For example, in aircraft engine design, one might need to compute a mesh for a given engine component, and pass this to a finite element computation that models the stresses in the part; this process alone might involve several different packages on different computers, in combination with some user interaction to guide the design process. It is not difficult to imagine more complex variations on this example – we might want to add thermodynamic models, fluid flow through the parts, etc, resulting in very complex and large workflows. Importantly, it is just as easy to envisage a design problem in computational chemistry in which one is computing binding energies between drugs and bio-molecules, and wanting to optimize the drug design to produce a more effective drug. Both of these examples involve the execution of multiple, often time consuming, computations in order to evaluate one design alternative, and highlight the need for a generic framework.

Recently, we designed a range of components for an existing workflow engine, Kepler [5][6][12][17], which allow a user to add "parameter sweep" operations to an existing workflow [2]. This extension makes it possible to run a workflow repeatedly, evaluating different input conditions. The new components generate combinations of parameter values up front, and these are streamed into the computational pipeline. Further extensions to Kepler make it possible to expose and exploit the dynamic parallelism that is created in such pipelines, making it possible to run the time consuming parts in parallel [3] on distributed high performance computers. For example, if there are sufficient resources available, it is possible to execute each design scenario in parallel, without modification of the original workflow. This system, called Nimrod/K, is based on the well-established Nimrod family of tools in combination with Kepler.

In this paper we extend our parameter sweep actors to support design optimization using a range of non-linear optimization algorithms. We design a further range of components that are suited to optimization, using an extensible technique that allows to not only add new techniques later, but also to mix and match different techniques using the basic building blocks. The system exploits parallelism between independent paths in the workflow, within a given search algorithm, and also by running multiple searches at the same time.

The paper proceeds with a general discussion of workflow engines, with a particular focus on Kepler, followed by a description of the Nimrod tool family. We then show how we have integrated the new optimization components into Kepler, producing a new system called Nimrod/OK. We illustrate the new solution with some real case studies, and briefly compare our solution to related projects.

2. Scientific Workflow Engines

Over the years we have developed expertise in massively parallel parameter sweep workflows, using the Nimrod family of tools [4]. Nimrod makes it very easy to run complex computational models whilst varying the input parameters – either across a complete sweep or as part of a search. Specifically, Nimrod contains tools that perform a complete parameter sweep across all possible combinations (Nimrod/G), or search using non-linear optimization algorithms (Nimrod/O) [22] or experimental design techniques (Nimrod/E) [21]. Importantly, the number of jobs, and thus the parallelism, can be varied at run time, and the Nimrod scheduler places tasks on the available resources at run time. However, Nimrod was not designed to execute arbitrary workflows. Thus, it is difficult to run sweeps *over* workflows, and workflows *containing* sweeps.

Scientific workflow engines, on the other hand, allow a user to perform complex calculations by combining a set of connected components, or *actors*. Actors represent a wide range of activities from computations, to access to distributed databases and scientific instruments. Typically, data *tokens* move between actors, and can be streamed from instruments and databases, through a range of computational processes. "Grid workflows" allow actors to run on distributed resources. They can be invoked in a variety of ways, facilitating virtual applications that span multiple organizations, data sources and computers.

Whilst many different scientific workflow tools have been built, in this paper we focus on Kepler [5][6][12][17]. However, many of the features in Kepler are similar to the other engines, and a good comparative review can be found in [27]. Kepler's engine orchestrates the execution of a workflow in a controlled and repeatable manner. It builds upon Ptolemy II [14], a Java-based software framework, including a graphical user interface called Vergil.

Ptolemy II is used for the modelling, simulation, and design of concurrent, real-time embedded systems. Kepler inherits a number of "directors" from Ptolemy, providing an extensive range of execution mechanisms.

In spite of their significant power, the Kepler runtime, and many other current workflow systems, do not support dynamic parallel execution of a workflow and its components. This means that users must explicitly program parallel and distributed computation into the workflow itself, complicating and often dwarfing the basic scientific processes or business logic. Typically this involves adding actors to execute graph nodes in parallel – either by replicating the workflow statically, or adding looping constructs that scatter and gather threads.

In another recent paper, we showed how to augment Kepler with a new orchestration mechanism called the Tagged Data Flow Architecture Director (TDA) [3]. This extends Kepler by providing powerful mechanisms for exposing and managing parallelism in the workflows, resulting a new tool called Nimrod/K.

In tagged token machines, tokens contain both a data field and a special tag field – or "colour", which is used to separate threads of execution. Importantly, an instruction fires when it has a token on each of its inputs that have the same colour values. Parallelism is implemented simply by creating tokens with different colours.

Tag values are manipulated by a set of special tag manipulation actors, although the tag flow implementation is usually hidden from workflow designers. Underlying a number of the common Kepler directors (PN and SDF) are FIFO queues located on each of the inputs on an actor. Normally, when multiple tokens arrive on an input port, they are queued and can be processed when the actor is available. Our TDA director follows the same procedure, but with a separate queue for tokens with different tags. This means that multiple tokens with the same tag value queue up, whereas tokens with different tag values can be consumed in parallel. Multiple actors cannot read from the same queue because they only read from queues with the same tag value assigned to them, and no two actors are given the same tag value. Importantly, this approach requires no changes to existing actors, which are usually unaware that they have been cloned. More details about the TDA are available in [3].

Nimrod/K provides an ideal platform for using workflows for optimization because it allows us to exploit parallelism within the computational pipeline, the optimization algorithm itself and between multiple optimizations. So, Nimrod/K extends the existing Nimrod family of tools significantly by adding Nimrod's parameter sweep technology to Kepler.

We have already added parameter sweeps actors to Nimrod/K [2] and shown that complete enumeration of parameter combinations, or partial combinations using experimental design techniques, can be performed in parallel. In this paper we show how we have extended this to support non-linear optimization operations that can be used in combination with existing workflows to form "Optimization Workflows".

3. Optimization using Nimrod/O

Before exploring the new optimization actors we have added to Kepler, we discuss how Nimrod/O can be used to perform automatic optimization using an existing computational model. There are two types of situation where the user of a computational model may wish to optimize some aspect of the output. The first occurs in system design where a simulation of a physical, financial or behavioural system is used to optimize some output, maximal performance or durability, say, or minimal cost or risk. The other situation is more common in computational modelling where the inputs are unknown but the outputs are required to fit some expected or observed values. This latter class, called inverse problems, are approached as optimizations, minimizing the difference between observed and desired outputs. Both types of problem have been successfully solved using Nimrod/O across a wide range of disciplines [13][22][20][1][23].

The types of optimization algorithms used for both classes of problems are iterative, starting from some point in the search space and normally converging to some (potentially local) optimum. Often the search space will have multiple local optima. Further, computational models often include noise in their outputs, caused by discretization errors for physical models, or finite sample errors for Monte-Carlo simulations. This noise may add spurious local minima to the search landscape. Consequently, multiple searches, from a variety of starting points, are commonly used to determine whether the optima found are global optimum, and because each search is independent, they can be executed in parallel.

A large variety of search algorithms have been employed for this task. The efficacy and efficiency of each depends on the nature of the objective landscape. For example, the BFGS algorithm [8] converges very rapidly if that landscape is sufficiently smooth. However, the method employs estimates of the downhill direction, which may

be seriously compromised if the surface is noisy. In such cases a more robust method such as the Nelder-Mead simplex algorithm may be more effective [19]. For models of systems with discontinuous objective functions, such as structural buckling [11], or allocation problems [8], downhill information may be tenuous, and biologically inspired methods such as genetic algorithms may be more suited.

When faced with these issues it is often difficult to choose the most appropriate algorithm in advance. Accordingly, we have found it is often necessary to try a variety of different algorithms, each with multiple starting points to address the problem of multiple local minima. Nimrod/O expedites such an approach by making it easy for a user to specify the use of multiple algorithms, and further, it executes these multiple searches in parallel to solve the problem as quickly as possible. Further, each iteration of an algorithm usually involves execution jobs in batches, depending on the potential for internal parallelisation of the algorithm. Nimrod/O will execute each batch in parallel where possible.

This parallelism has led to modification of search algorithms, and reassessment of their efficiency [22]. If the user is concerned only with the wall clock time for convergence of an algorithm then the number of batches, rather than the number of jobs, becomes the salient measure of efficiency. Our implementation of various algorithms often takes advantage of this fact. For example, in the venerable Hooke and Jeeves algorithm [24], the "exploration phase" varies each coordinate of the current point up or down until an improvement is detected, a serial search evaluation one point at a time. If n is the dimensionality of the search space then there are potentially 3^n points that might be reached. Our implementation has an option called "parallel Hooke and Jeeves" that evaluates all these possible points concurrently. Such a variant may make the method more competitive with alternative algorithms when judged by the number of batches required.

4. Nimrod/OK – Optimization Workflows

Optimization algorithms may themselves be viewed as workflows, usually involving repetitive looping so that results are passed from one iteration to the next. But optimization codes are traditionally monolithic with various components tailored to a particular algorithm. Implementing the functionality of Nimrod/O within the Kepler workflow engine exposes the components of the optimization process, giving the user a clear view of the data flows and facilitating substitution of these components. The result is an extension of Kepler that we have termed Nimrod/OK.

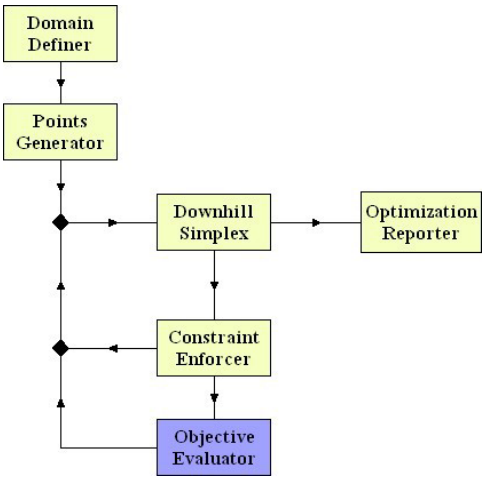

Figure 1 – optimization as a workflow

Figure 1 shows a possible workflow associated with the commonly used the Nelder-Mead Simplex optimization algorithm [19]. First, the domain of the search is defined by specification of the input parameters, their nature (floats, integers or text parameters) and ranges. Within this domain the next component will select starting points for

the multiple optimizations. Each starting point will begin a simplex search. The search algorithm generates sets of points that are sent for evaluation, a "batch of jobs" in Nimrod/O's terminology. Running the models represented by these jobs produces numerical results that are used to decide the next generation of points. This cycle continues until some convergence criterion has been achieved whereupon a final point is forwarded as the result of the search. Figure 1 also shows a component that evaluates constraint conditions imposed by the user. This calculates the margin by which a search point violates any constraints, and imposes penalties accordingly. If the constraint is a "hard" one then the point is completely forbidden, obviating the need for model evaluation.

Implementing such optimizations as a workflow, the fundamental decision is whether to treat the components as persistent *stateful* or *stateless* (functional) actors. Actor state includes the memory of the domain, for example, the starting point and other information representing the state of the optimization. The alternative is not to store the state in the actors, but to circulate such information around the workflow with the point sent for evaluation. We have chosen the latter as the more flexible solution. It enables modification of these settings by the insertion of monitor/control actors. It allows workflows to be diverted into novel paths, for example the results of one algorithm may, after some number of iterations, be sent to a different algorithm. Our mechanism for forwarding this state information is the Tagged Data Flow Architecture of Nimrod/K. (This tags the job information with a Java object that contains all the information required.)

Figure 2 shows a sample Kepler workflow in Nimrod/OK. The Define search space actor and the Select search space points actor correspond to the Domain Definer and Points Generator of Figure 1. The user enters appropriate specifications into these actors by setting the actors' parameters. In this implementation the initial points may be selected randomly, or be regularly spaced throughout the domain, or be specified directly.

The points selected are forwarded to a Simplex Optimization actor, which initiates several simplex searches, one for each starting point. These searches produce batches of jobs that are sent to the evaluation actor, in this case one that runs a MATLAB computational model. The results from the MATLAB actor are then sent to an actor that computes the RMS error, illustrating that optimization runs can be performed across pipelines of computations. Settings for the optimization, such as the precise variant of the algorithm, the convergence criterion, and the action to take if a job fails, may also be entered as optimization actor parameters. Conveniently, these settings may be modified while the optimization proceeds. When convergence is obtained, the final optima are forwarded to the Display actor. Statistics on the optimizations are sent to Display2.

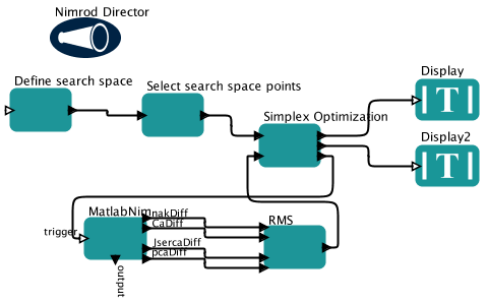

Figure 2 – use of new optimization actors

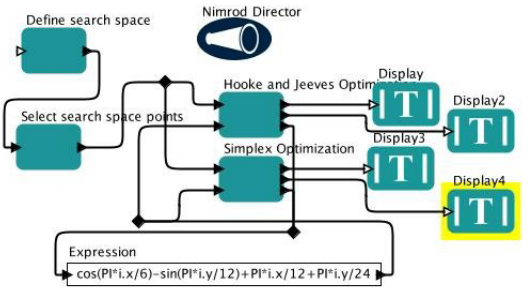

Figure 3 – use of multiple optimization algorithms

A more complex optimization is shown in Figure 3. After the selection of the initial points, the workflow branches into two separate search algorithms: simplex and Hooke-Jeeves. Under the Nimrod director these optimizations will (where resources allow) run concurrently. Both actors forward jobs to the Kepler expression actor, which supplies the objective function in this example.

The workflow shown Figure 4 illustrates the flexibility of Nimrod/OK, performing optimizations within a wider parameter sweep. The sweep actor (previously described in [3]) performs a sweep over one of the input parameters; each value of that parameter then spawns a separate search. (Such an approach is convenient where some of the parameters are discrete values.) Here the search algorithm is a subdivision search that repeatedly refines the domain

as indicated by evaluation at a grid of points. This is applied for a few iterations. The best point found is then diverted to a simplex search.

It is worth highlighting the significant differences between the new functionality discussed here and our previous implementations of Nimrod/O. Nimrod/O optimization algorithms are self contained, and can only be altered by changing the code of Nimrod/O itself. Importantly, they cannot be combined with each other. Nimrod/OK exposes key components in each optimization algorithm, allowing them to be reconfigured and used in novel ways (as shown in Figure 4). This allows optimization algorithms to share common functions (like defining the search space, detecting start points, etc), and the sharing is obvious by examining the workflow. Further, new optimization algorithms (or components) can be integrated more easily because they are added as new actors, rather than by modifying the existing code base. Finally, and perhaps most importantly, Nimrod/O does not allow pipelines of computations where each step might run on a different platform. In contrast, Nimrod/OK does this because it leverages an existing workflow engine, Kepler, plus our additions in Nimrod/K that exploit parallelism transparently. Whilst our examples only illustrate small pipelines, clearly, they can be arbitrarily complex.

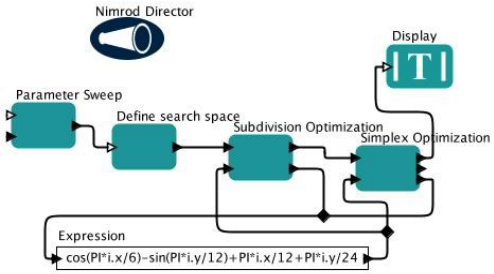

Figure 4 – optimizations within a sweep

5. Case Study

Here we further illustrate the new optimization actors by solving an inverse problem in cardiac electro-physiology research. The electrical activity of the heart is based on cycles of ion transfer via cell surface membrane. There has been much research on developing robust and accurate models of myocyte behaviour so that in-silico experiments can be performed on complete cardiac systems. The research has significance from basic science through to therapeutic drug design.

The Shannon-Bers model [25] is a system of coupled differential equations that mimic the ionic flows in rabbit heart cells. In [18], the authors have extended this model to investigate the effects of addition of metabolic factors on the ventricular myocyte excitation-contraction coupling. The model is implemented using MATLAB. In order to validate the model, some final outputs were compared against experimentally known values. This involved exploring a range of numbers for nine input metabolic constants and selecting the combination which produced the closest output to the desired value.

This is a typical inverse problem, approached by minimising the sum of the squares of the differences between model values and experimental ones. Our first demonstration repeats this optimization using Nimrod/OK with the workflow shown in Figure 2. It demonstrates that the system can utilise proprietary software.

A nine dimensional optimization may be computationally expensive. However, a fractional factorial investigation using Nimrod/E into the effects of these nine inputs suggested that they fell into four mutually exclusive groups that showed no interaction between groups [7]. This indicated that four separate optimizations, one over each subgroup, would be more efficient. A second experiment, shown in Figure 5, ran these four searches as a single workflow. To achieve this, the Define search space actors were customized with some of parameters specified as constants, allowing only the active parameters to vary. The searches are 2, 1, 3 and 3-dimensional respectively.

Again, we highlight that it would not be possible to implement the workflow shown in Figure 5 with original Nimrod/O tool. Instead we would have had to execute 4 independent Nimrod/O runs with significant additional complexity.

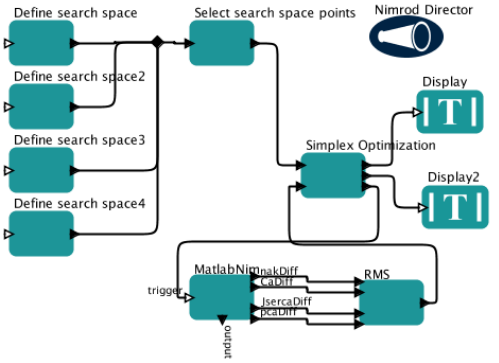

Figure 5 – partition of the optimization domain

6. Results

Experiment 1 performed simplex searches over the nine-dimensional space, using the workflow shown by Figure 2. The computations are performed on a distributed Grid of Linux clusters, as shown in Table 1.

Machine	# Cores	Hardware	Location
East	160	Intel(R) Xeon(R) E5310 @ 1.60GHz	Monash University, Melbourne
	64	Intel(R) Xeon(R) 5160 @ 3.00GHz	
West	160	Intel(R) Xeon(R) E5310 @ 1.60GHz	Deakin University, Geelong
South	160	Intel(R) Xeon(R) E5310 @ 1.60GHz	RMIT, Melbourne

Table 1 – Properties of Grid Testbed

128 starting points were randomly selected within the domain. Table 2 summarises the results of these searches, showing the best, worst and average search results. We also show the number of jobs and batches required to achieve the best and worst results, and the average number of jobs and batches. The best search gave a considerably better (smaller) result than the average, at the expense of more computational effort as shown by the numbers of jobs and batches of jobs. This shows the utility of multiple searches.

Job executions averaged 3m 21s, but with substantial parallelism the full experiment was completed in 7h 37m. Figure 6 plots the number of jobs executing over the duration of the experiment. Initially the simplex method evaluates the objective at the vertices of the starting simplex. In this case each of the simplexes has ten such vertices, so the experiment began with 1280 jobs. Thereafter, most iterations require just four new evaluations, so the load dropped to 512 jobs, except for the occasional peak where a new simplex was required. As searches completed the load dropped in stages; the final longest running optimization required only four processors but dominated the experiment wall clock time.

Index	Objective	Jobs	Batches
Min	0.00042	154	37
Average	0.0015	142	35
Worst	0.0038	34	7

Table 2 – results for searches over the full domain

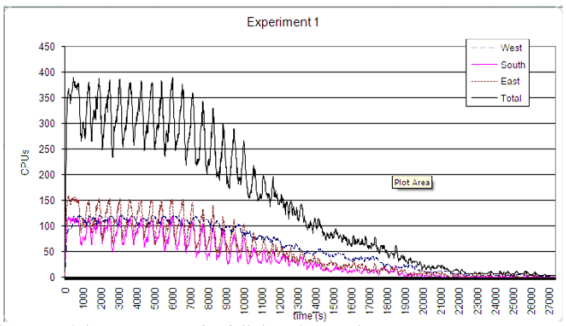

Figure 6 – job concurrency for full domain search

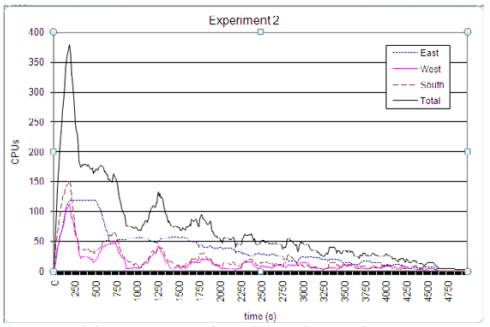

Figure 7 – job concurrency for subdomain searches

The second experiment used the configuration shown in Figure 5. This time, 32 optimizations were used for each of the four domains, with randomly selected starting points. Although this generated 128 parallel searches, these had low dimensionality and hence required smaller initial batches.

Each parameter, in the optimizations where it was varied, attained a best value close to that for the nine-dimensional search. However the lower dimensionality promoted faster convergence as shown in Figure 7. This is demonstrated by Figure 8, which superimposes a plot of the concurrency of Experiment 2 on that for Experiment 1. Execution wall clock time was 1h 23m, a substantial reduction over Experiment 1.

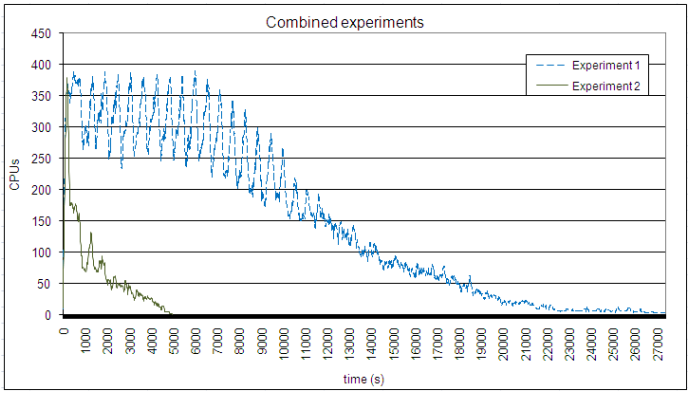

Figure 8 – job concurrency for both experiments

7. Related work

Our work has many similarities with that of Crick et al. [10], which recognises the significant advantages in combining optimization tools with existing "generic" workflow tools. One difference is the choice of workflow engine, Kepler as opposed to Taverna [15], and in particular the extensions in Nimrod/K that support dynamic parallelism in the workflow. This means that optimization algorithms that have internal parallelism, such as Simplex and Genetic Algorithms, can exploit multiple processes transparently. Further, it is simple to combine a parameter sweep actor with an optimization run to perform multiple searches concurrently. But more importantly, we have developed a library of different optimization components that can be mixed in powerful ways.

Our work also has a lot in common with Geodise, where the goal is to "expose optimization services in a flexible, generic interface that can be easily integrated into various environments and used to compose different optimization workflows." [26][16]. In Geodise, optimization algorithms are exposed as Web services, which can be invoked by a workflow system, or any other programming language that can execute Web service calls. In our case we have embedded our work in an existing workflow engine, Kepler, and have built Kepler actors that perform the

optimization functions. We are exploring building hybrid optimization algorithms by combining different actors, and this is presumably possible within Geodise, although we have not seen an example of such a workflow. As discussed above, the Nimrod/K engine allows us to exploit parallelism in the workflow transparently.

8. Conclusion

Nimrod/OK recasts then traditional tasks of optimizing the results of computational models within the workflow paradigm. This adds flexibility to the design of such experiments as demonstrated by the various workflows shown where various search methods are connected in novel configurations. This work leverages the wide offering of workflow actors supplied by Kepler. It also builds upon the Nimrod/K framework allowing concurrent execution of the computational models, utilizing multiple CPUs on a local machine or the resources of a computational Grid.

Whilst the implementation shown here uses Kepler, the techniques could be applied to any number of workflow systems. The factoring of optimization components is independent of the workflow tool, and these could be mapped into other systems such as Taverna and Triana. However, Nimrod/K is somewhat unique in the way it exposes parallelism, and this would need to be considered if the optimization algorithms were added to other workflow engines.

This paper has demonstrated the usefulness of the approach on an important computational biology application, calibration of the input metabolic constants in a computational model of rabbit ventricular myocytes.

In future developments, we plan to expand the range of optimization algorithms implemented, and then explore novel combinations of these. For example, combining some greedy local search heuristics in an evolutionary algorithm, such as a genetic algorithm, might improve the overall search. Deft choice of components should make this possible.

Acknowledgements

This project is supported by the Australian Research Council under the Discovery grant scheme We acknowledge our colleagues in the MeSsAGE Lab. at Monash University and the Cardiac Mechanics Lab. at UCSD. We also acknowledge the US NSF funded PRIME project, (supporting Amirriazi and Chitters) and thank colleagues, in particular Peter Arzberger, for their strong support.

References

1 D. Abramson, A. Lewis, T. Peachey, C. Fletcher, An Automatic Design Optimization Tool and its Application to Computational Fluid Dynamics, SuperComputing 2001, Denver, Nov 2001.

2 D. Abramson, B. Bethwaite, C. Enticott, S. Garic and T. Peachey, Parameter Space Exploration using Scientific Workflows, ICCS 2009, SU Center for Computation & Technology, Baton Rouge, Louisiana, U.S.A, May 25-27, 2009.

3 D. Abramson, C. Enticott and I. Altinas, Nimrod/K: Towards Massively Parallel Dynamic Grid Workflows, IEEE Supercomputing 2008, Austin, Texas, November 2008.

4 D. Abramson, J. Giddy, and L. Kotler, High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid, In Int'l. Parallel and Distributed Processing Symposium (IPDPS), Cancun, Mexico, May 2000. www.messagelab.monash.edu.au/nimrod/

5 I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher and S. Mock, Kepler: Towards a Grid-Enabled System for Scientific Workflows, in the Workflow in Grid Systems Workshop in GGF10 - The 10th Global Grid Forum, Berlin, March 2004.

6 I. Altintas, A. Birnbaum, K. Baldridge, W. Sudholt, M. Miller, C. Amoreira, Y. Potier and B. Ludaescher, A Framework for the Design and Reuse of Grid Workflows, Intl. Workshop on Scientific Applications on Grid Computing (SAG'04), LNCS 3458, Springer, 2005

7 S. Amirriazi, S. Chang, T. Peachey, D. Abramson and A. Michailova, Optimizing Cardiac Excitation-Metabolic Model By Using Parallel Grid Computing, Biophysics 2008, Long Beach, California, February 2008

8 C. G. Broyden, The Convergence of a Class of Double-rank Minimization Algorithms, Journal of the Institute of Mathematics and Its Applications, 6 (1970) 76-90.

9 Y.-H. Changa and Y.-C. Houb, Dynamic programming decision path encoding of genetic algorithms for production allocation problems, Computers & Industrial Engineering, 54 (2008) 53-65.

10 T. Crick, P. Dunning, H. Kim and J. Padget, Engineering design optimization using services and workflows, Phil. Trans. R. Soc. A 367 (2009) 2741-2751.

11 B. G. Falzon and D. Hitchings, An Introduction to Modelling Buckling and Collapse, NAFEMS, 2006.

12 http://kepler-project.org/ , accessed 10 March 2010.

13 http://messagelab.monash.edu.au/EScienceApplications/ , accessed 10 March 2010.

14 http://ptolemy.eecs.berkeley.edu/ptolemyII/ptIIfaq.htm , accessed 10 March 2010.

15 http://www.taverna.org.uk/ , accessed 10 March 2010.

16 http://www.geodise.org/ , accessed 10 March 2010.

17 B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao and Y. Zhao. Scientific Workflow Management and the Kepler System, Concurrency and Computation: Practice & Experience, Special Issue on Workflow in Grid Systems, 18 (2006) 1039-1065.

18 A. Michailova, W. Lorentz, A. McCulloch, Modelling Transmural Heterogeneity of KATP current in Rabbit Ventricular Myocytes, AJP-Cell Physiology, 293 (2007), C542-C557.

19 J. A. Nelder and R. Mead, A simplex method for function minimization, Computer Journal, 7 (1965) 308-313.

20 T. C. Peachey and C. M. Enticott, Determination of the Best Constant in an Inequality of Hardy, Littlewood and Polya, Experimental Mathematics, 15 (2006) 43-50.

21 T. C. Peachey, N. T. Diamond, D. A. Abramson, W. Sudholt, A. Michailova, S. Amirriazi, Fractional Factorial Design for Parameter Sweep Experiments using Nimrod/E, Journal of Scientific Programming, 16 (2008) 217-230.

22 T. Peachey, D. Abramson and A. Lewis, Model Optimization and Parameter Estimation with Nimrod/O, The International Conference on Computational Science, May 28-31, University of Reading 2006.

23 T. Peachey, D. Abramson, A. Lewis, D. Kurniawan and R. Jones. Optimization using Nimrod/O and its Application to Robust Mechanical Design, PPAM 2003, Czestochowa, Poland, Lecture Notes in Computer Science, 3019 (2004) 730 – 737.

24 R. Hooke and T. A. Jeeves, Direct search solution of numerical and statistical problems, Journal of the ACM, 8 (1961) 212 - 229.

25 T. R. Shannon, F. Wang, J. Puglisi, C. Weber and D. M. Bers, A Mathematical Treatment of Integrated Ca Dynamics within the Ventricular Myocyte, Biophysical Journal, 87 (2004), 3351-3371.

26 G. Xue, W. Song, S.J. Cox and A. J. Keane, Numerical Optimization as Grid Services for Engineering Design, Journal of Grid Computing, 2 (2004) 223-238.

27 J. Yu J., and R. Buyya, A Taxonomy of Workflow Management Systems for Grid Computing, Journal of Grid Computing, 3 (2005) 171-200.